1.3.4.3 Flexibilidade

Embora, no estado de arte, a falta de padronização das linguagens de programação torne a produção de programas absolutamente portáteis uma utopia que parece ainda distante, o uso da PE ajuda a transportar programas de máquina a máquina, seja através da melhoria da documentação, seja por restringir a sintaxe, simplificando a tradução manual ou automática.

Freqüentemente, na vida útil dos programas, mudam suas especificações: algumas vezes a experiência adquirida com seu uso realimenta uma reespecificação, outras descobrem-se aplicações às quais o programa pode ser adaptado lucrativamente.

Pelas mesmas razões por que contribui para tornar os programas portáteis, a PE contribui para torná-los adaptáveis. Ou, pode-se dizer, combinando ambos os conceitos, para torná-los mais flexíveis.

1.3.4.4 Importância da Documentação

Dificilmente se encontrará um programador que ponha em dúvida a importância da boa documentação dos programas: todos sabem que boa documentação facilita a vida de todos: de quem programa, de quem modifica, de quem usa e de quem administra. Apesar disto, o mundo da programação está cheio de programadores que, de código escrito, testado e tudo, só podem responder a quem lhes cobra a documentação com um "devo, não nego, pagarei quando puder", pronunciado compungidamente.

Por que a aversão de tantos programadores (freqüentemente, os mais brilhantes) a documentar? Na programação tradicional, a documentação consiste habitualmente em fluxogramas e comentários, que não são indispensáveis à execução do programa e podem, por isto mesmo, ser feitos *a posteriori*. Por isto mesmo eles trazem para muitos programadores a mesma gratificação que a lavagem dos pratos depois de um opíparo festim.

A PE ataca o problema propondo métodos de desenvolvimento que produzem simultaneamente o programa e sua documentação, ou melhor, procuram tornar código e documentação aspectos indivisíveis de um mesmo produto, que ficam forçosamente prontos ao mesmo tempo. Dirá o leitor atento: ora, isto não é mais que uma regra de bom senso, aplicável não só à construção de programas, mas de qualquer projeto. É isto mesmo. O que bem ilustra a essência da PE, que é a sistematização do bom senso na atividade de programar.

1.3.4.5 Fluxo de Controle em Algoritmos

Nesta seção vamos discutir os problemas do fluxo de controle em algoritmos, em particular a problemática do comando GOTO, justificando assim o uso exclusivo de um número restrito de estruturas básicas de controle na programação estruturada.

Em primeiro lugar, um algoritmo é um *texto estático*. Ele existe numa folha de papel, escrita e geralmente lida de cima para baixo. Para produzir os resultados desejados, esse algoritmo deve ser executado; temos então um *processo dinâmico*. O conceito que relaciona esses

dois aspectos é o de *fluxo de controle*, que determina em cada passo da execução qual é o próximo comando a ser executado.

A ordem de execução dos comandos geralmente é bem diferente da sua ordem na folha de papel. Não obstante tentamos (e precisamos) compreender a ordem das ações que ocorrem durante a execução, baseando-nos simplesmente na descrição estática do texto do algoritmo. O problema se complica pelo fato que a ordem de execução dos comandos em geral muda de uma execução para outra, dependendo dos dados de entrada, ou seja, que temos que considerar de fato não somente um processo, mas toda uma classe de diferentes processos possíveis que o algoritmo pode evocar.

Entender um algoritmo (a *lógica* de um algoritmo) significa compreender (visualizar) essa classe de processos de execução. Analogamente, afirmar que um programa é correto quer dizer essencialmente que todas suas execuções possíveis produzem resultados corretos. Vemos então que, para ler ou entender um programa, lidamos fisicamente com um texto, mas temos em mente processos.

A maior dificuldade que o programador tem que enfrentar é então a *distância conceitual* que separa a representação estática de um algoritmo do (s) processo (s) dinâmico (s) de sua execução.

A dificuldade fundamental do comando GOTO é que ele obriga o leitor a ler um algoritmo em forma pouco natural, ou seja, "pulando" mentalmente para frente e para trás no programa. Além disso, para cada GOTO condicional ele deve ainda, lembrar de onde ele "pulou", para poder retomar ali a análise do caso alternativo (da condição do GOTO ser *falsa*).

Vejamos como exemplo, ainda muito simples, um trecho de programa em FORTRAN para mostrar a problemática:

```
      ⋮
      IF (A .GT. 15) GOTO 3
      IF (A .GT. 10) GOTO 2
      IF (A .GT.  5) GOTO 1
      X = 0
      Y = 0
      GOTO 4
    1 X = X * 2
      Y = Y + 1
      GOTO 4
    2 X = X * 3
      Y = Y + 2
      GOTO 4
    3 X = X * 4
      Y = Y + 3
    4 CONTINUE
      ⋮
```